PRECIOUS: Approximate Real-Time Computing in MLC-MRAM based Heterogeneous CMPs

Sangeet Saha, Shounak Chakraborty, Sukarn Agarwal, Magnus Själander, and Klaus McDonald-Maier

Abstract—Enhancing quality of service (QoS) in approximate-computing (AC) based real-time systems, without violating power limits is becoming increasingly challenging due to contradictory constraints, i.e., power consumption and time criticality, as multicore computing platforms are becoming heterogeneous. To fulfill these constraints and optimise system QoS, AC tasks should be judiciously mapped on such platforms. However, prior approaches rarely considered the problem of AC task deployment on heterogeneous platforms. Moreover, the majority of prior approaches typically neglect the runtime architectural phenomena, which can be accounted for along with the approximation tolerance of the applications to enhance the QoS. We present *PRECIOUS*, a novel hybrid offline-online approach that first *schedules AC real-time* tasks on a *heterogeneous multicore* with an objective to maximise QoS and determines the appropriate cluster for each task constrained by a system-wide power limit, deadline, and task-dependency. At runtime, *PRECIOUS* introduces novel architectural techniques for the AC tasks, where tasks are executed on a heterogeneous platform equipped with *multilevel-cell (MLC)-MRAM* based last-level cache to improve energy efficiency and performance by prudentially leveraging storage density of MLC-MRAM while ameliorating associated high write latency and write energy. Our novel block management for the MLC-MRAM cache further improves performance of the system, which we exploit opportunistically to enhance system QoS, and turn off processor cores during the dynamically generated slacks. *PRECIOUS-Offline* achieves up to 76% QoS for a specific task-set, surpassing prior art, whereas *PRECIOUS-Online* enhances QoS by 9.0% by reducing cache miss-rate by 19% on a 64-core heterogeneous system without incurring any energy overhead over a conventional MRAM based cache design.

Index Terms—Real-time Approximate Computing, QoS, Energy Efficiency, Caches, Heterogeneous Systems, MLC-MRAM

1 Introduction

In approximate time-critical systems, delivering an approximated result before a deadline is favoured over an accurate but late one [29], a common scenario in multimedia processing, mobile target tracking, real-time heuristic search, information gathering, and control systems [4]. For instance, lower-quality video frames are preferred to missing ones, and an estimated target location within the time limit is better than a precise one arriving late. Applications in these domains are often modeled as precedence constrained task graphs (PTGs), where nodes represent tasks with dependencies indicated by edges. Each task comprises a mandatory part, which must finish by the deadline to ensure a minimum acceptable Quality of Service (QoS), and one or more optional parts [9], [28], [31]. Executing the optional part(s) partially or fully, depending on available resources, enhances the initial QoS within the deadline. Higher QoS levels demand more computation, resulting in longer execution times and increased power consumption. Consequently, achieving improved QoS, enhanced power efficiency, and on-time task completion often presents conflicting objectives.

Time-critical systems modeled as PTG, conventionally implemented on homogeneous multiprocessors, are increasingly scheduled on heterogeneous platforms (HE) [14], [27] to meet performance, reliability, and power constraints. On HE systems, task resource needs and execution times vary by core. *A critical scheduling challenge involves maximising system-level QoS for an AC real-time PTG with multi-level QoS tasks on a heterogeneous multicore by optimally assigning QoS levels, processor cores, and start times while adhering to timing, power, precedence, and resource constraints.* While offline scheduling is preferred for timing predictability in such systems [36], its static nature can compromise runtime performance if architectural characteristics are not taken into account [10]. Runtime architectural optimisations, particularly for last-level caches (LLCs) [11], [33], can be leveraged to create QoS-aware and power-efficient real-time platforms.

Over the last decade, non-volatile memories (NVMs) like MRAM/STT-RAM, PCM, and flash have been investigated to tackle SRAM cache leakage power [1]. MRAM shows promise as an SRAM replacement in caches due to its low leakage, high endurance, and comparable read times [2]. Multilevel-cell (MLC)-MRAM further doubles storage density, increasing cache capacity, which is highly beneficial for demanding time-critical workloads. In this paper, we propose *PRECIOUS*, which *employs an MLC MRAM-based LLC architecture on a heterogeneous multicore platform to enhance both energy efficiency and storage density, thereby improving the QoS of approximate real-time computations—a notable advancement over conventional SRAM or single-level cell (SLC) MRAM cache designs.*

In *PRECIOUS*, we introduce a hybrid heterogeneous task-scheduling technique for approximate real-time tasks. *PRECIOUS-Offline* initially generates an offline schedule to maximise QoS, forming a foundation for the online phase.

S.Saha and K. McDonald-Maier are with the Embedded and Intelligent Systems Lab, University of Essex, Colchester, UK. S. Chakraborty is with the Department of Computer Science, Durham University, UK. S. Agarwal is with the EECS Department, Indian Institute of Science Education and Research Bhopal, India. M. Själander is with the Department of Computer Science, Norwegian University of Science and Technology, Trondheim, Norway.
e-mail: (sangeet.saha@essex.ac.uk, shounak.chakraborty@durham.ac.uk, sukarn@iiserb.ac.in, magnus.sjalander@ntnu.no, kdm@essex.ac.uk).

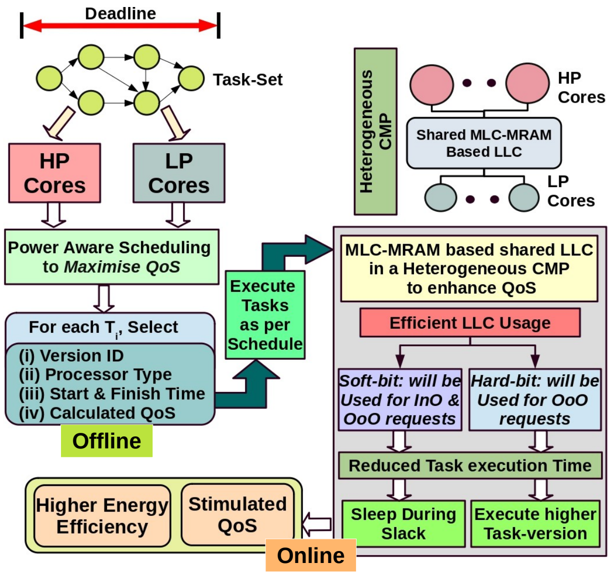
Deadline
Task-Set
HP Cores
LP Cores
Power Aware Scheduling to Maximise QoS
For each T_i, Select
(i) Version ID
(ii) Processor Type
(iii) Start & Finish Time
(iv) Calculated QoS
Offline
Execute Tasks as per Schedule
Heterogeneous CMP
HP Cores
Shared MLC-MRAM Based LLC
LP Cores
MLC-MRAM based shared LLC in a Heterogeneous CMP to enhance QoS
Efficient LLC Usage
Soft-bit: will be Used for InO & OoO requests
Hard-bit: will be Used for OoO requests
Reduced Task execution Time
Sleep During Slack
Execute higher Task-version
Higher Energy Efficiency
Stimulated QoS
Online

Fig. 1: *PRECIOUS*: Process Overview

PRECIOUS-Online then dynamically refines this schedule at runtime to significantly boost QoS by considering runtime characteristics and optimizing MLC-MRAM cache utilisation without incurring performance or energy penalties. Our dependent task set is represented by a PTG (shown in Sec. 2) and scheduled on a heterogeneous multicore processor, where each task offers different QoS levels via multiple versions. The entire *PRECIOUS* concept is illustrated in Figure 1, with the offline part on the left and the online part on the right. The major contributions of *PRECIOUS* can be listed as follows:

1) With an objective to maximise QoS, *PRECIOUS-Offline* presents a power-constrained, optimal, task-scheduling strategy devised for a set of approximated real-time dependent tasks on a heterogeneous system, allocating each task at either of two different clusters of cores (high-performance cores, represented as C_H, and low-performance cores, represented as C_L) (Sec. 3).
2) At runtime, *PRECIOUS-Online* prudentially leverages higher storage density of MLC-MRAM based LLC to enhance performance of individual tasks by accumulating more live LLC blocks on-chip while mitigating higher write cost of MLC-MRAM by employing novel block management mechanism (Sec. 4).
3) The improved performance reduces the execution length of each task, which is further traded off either to enhance system level QoS by opportunistically executing task-version having higher QoS level, if available, or to improve energy efficiency by turning off the cluster(s) during online slacks (Sec. 4).

Surpassing prior art, *PRECIOUS-Offline* achieves up to 76% QoS, and *PRECIOUS-Online* further improves this by 9.0% through a 19% reduction in cache miss-rate on a 64-core heterogeneous system, without increasing energy consumption compared to a conventional SLC-MRAM LLC on a 64-core HE CMP with an MLC-MRAM LLC (Sec. 5). To the best of our knowledge, *PRECIOUS* is the first work to maximise QoS for an approximate dependent time-critical task set by using a novel scheduling strategy on a heterogeneous multicore processor. It further enhances QoS through a novel architectural technique that judiciously utilises the increased storage density of an MLC-MRAM-based LLC while mitigating higher write costs, without incurring performance or energy overhead.

2 System Model and Assumptions

By considering the characteristics of an existing HE platform (e.g., ARM big.LITTLE) [25], we introduce the concept of cluster. For instance, our platform consists of two HE clusters: the high performance (HP) cluster (denoted as C_H) and the low performance (LP) cluster (denoted as C_L), where all cores from the same cluster run at the same frequency, f_c. Note that the processor cores belonging to the same cluster are homogeneous. We represent our application as a PTG (see Figure 2), $G = (T, E)$, where T is a set of tasks ($T = T_i | 1 \leq i \leq n$) and E is a set of directed edges ($E = \{\langle T_i, T_j\rangle \mid 1 \leq i, j \leq n; i \neq j\}$), indicating the task-dependency or precedence relations between a distinct pair of tasks. As a real-time application, G needs to be executed within a given deadline, D_{PTG}, by executing all of its associated tasks (T_i). We further assume that, T_i can have k_i different versions, $T_i = T_i^1, T_i^2, ..., T_i^{k_i}$, those are distinct by their respective execution lengths (O_i), which can be denoted as $O_i^1, O_i^2, ..., O_i^{k_i}$, while O_i^p offers higher QoS than O_i^q , if $p > q$. For each of the optional parts of a task (O_i), there exists a separate executable module, which is executed after executing the mandatory portion (M_i) of the respective task, T_i. The execution length of the v-th version of task T_i (len_i^v) can be defined as: $len_i^v = M_i + O_i^v$. Note that, len_i^v includes the cycles needed for accessing LLC, which we obtain by executing an individual task for a specific configuration. We next define result-accuracy/QoS Acc_i^v of T_i^v as the executed optional portion of the task, O_i (i.e., $Acc_i = O_i^v$). Hence, the overall system level QoS is now defined as the summation of the executed cycles of O_i^v for all the tasks [9], which can be represented as: $QoS = \sum_{i=1}^{n} O_i^v \mid T_i = T_i^v$. To represent the heterogeneity among the processor cores in different clusters, we introduce a factor $\gamma_{i,c} \in (0, 1]$ [23], which indicates the efficiency factor of HE processors executing task T_i. Therefore, the execution time of task T_i executing with version v at c^{th} cluster can be represented as $ET(i, v, Cl_c)$ is $\frac{len_i^v}{\gamma_{i,c} \times f_c}$.

TABLE 1: Parameters and their values, for example task-set

Tasks	C_L				C_H			
	M_i	O_i	Pow_i	$\gamma_{i,L}$	M_i	O_i	Pow_i	$\gamma_{i,H}$
T_1^1	4	2	10	0.5	4	2	13	0.7
T_2^1	10	5	20	0.4	10	5	26	0.6
T_2^2	10	8	22	0.4	10	8	29	0.6
T_2^3	10	10	24	0.4	10	10	30	0.6
T_3^1	10	2	10	0.5	10	2	12	0.6
T_3^2	10	4	12	0.5	10	4	14	0.6
T_3^3	10	6	15	0.5	10	6	17	0.6
T_4^1	28	16	15	0.4	28	16	18	0.8
T_4^2	28	20	18	0.4	28	20	21	0.8
T_5^1	8	2	20	0.5	8	2	25	0.7
T_5^2	8	4	24	0.5	8	4	28	0.7
T_5^3	8	5	27	0.5	8	5	31	0.7
T_6^1	10	3	10	0.5	10	3	12	0.7
T_6^2	10	6	12	0.5	10	6	14	0.7
T_6^3	10	9	14	0.5	10	9	16	0.7

3 *PRECIOUS-Offline*

With an objective to maximise the QoS without violating the system-wide constraints, *PRECIOUS-Offline* introduces

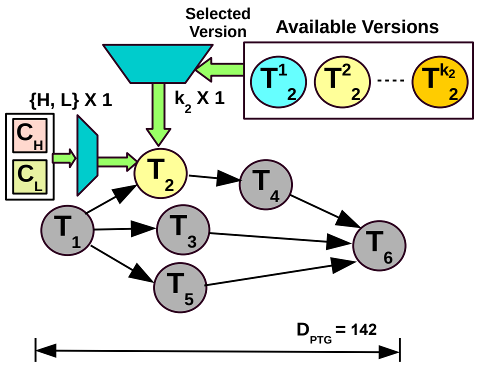

Fig. 2: Precedence Task Graph (PTG).

a scheduling strategy to allocate tasks to the processor cores in an HE platform, detailed in Sec. 3.1.

3.1 Scheduling Approach

At first, we present our scheduling strategy based on Integer Linear Programming (ILP), for which, we will introduce an integer decision variable $S_i \in \mathbb{Z}^+$ to capture start time of each task T_i, where $\mathbb{Z}^+$ denotes the set of positive integers. We next define a binary decision variable, Z_{i,v,cl_c}, where, $i = 1, 2, ..., n$; $cl_c \in \{C_H, C_L\}$; and v denotes the task's version. If T_i executes on cluster cl_c with version v, then $Z_{i,v,cl_c} = 1$, otherwise 0. We define another binary variable Y_{ij}, where $Y_{ij} = 1$, if task T_i starts before T_j, else 0, defines the precedence order for mutually independent tasks.

To model our scheduling strategy, the constraints on our decision variable are now represented as follows:

1) Each task T_i is assigned to exactly one cluster with one particular version (v):

$$\forall i \in [1, n] | \sum_{v=1}^{k_i} \sum_{cl_c \in \{C_L, C_H\}} Z_{i,v,cl_c} = 1 \tag{1}$$

2) The application $\mathcal{A}$ must meet its end-to-end absolute deadline D_{PTG}. Hence, the sink node T_n should be finished by D_{PTG}, which we represent as follows:

$$S_n + \sum_{v=1}^{K_i} \sum_{cl_c \in \{C_L, C_H\}} (ET(n, v, cl_c) \times Z_{n,v,cl_c}) - 1 \leq D_{PTG} \tag{2}$$

3) Precedence constraint between the tasks must also be satisfied. The execution of T_j should commence only after the completion of its predecessor T_i which can be formulated as: $\forall(\langle T_i, T_j \rangle) \in E$,

$$S_i + \sum_{v=1}^{k_i} \sum_{cl_c \in \{C_L, C_H\}} [ET(i, v, cl_c) \times Z_{i,v,cl_c}] \leq S_j \tag{3}$$

4) No two tasks can be assigned to the same cluster, such that their execution instances overlap. However, for the task pair that shares a precedence relationship, this non-overlapping constraint is imposed on them by default. To avoid overlapping for the rest of the mutually independent task pairs, the following inequalities need to be satisfied: $\forall(\langle T_i, T_j \rangle) \notin E$, where $i \neq j$,

$$Y_{ij} + Y_{ji} = 1 \tag{4}$$

$$S_i + \sum_{v=1}^{k_i} \sum_{cl_c \in \{C_L, C_H\}} [ET(i, v, cl_c) \times Z_{i,v,cl_c}] - (1 - Y_{ij}) \times M \leq S_j \tag{5}$$

$$S_j + \sum_{v=1}^{k_i} \sum_{cl_c \in \{C_L, C_H\}} [ET(j, v, cl_c) \times Z_{j,v,cl_c}] - (Y_{ij} \times M) \leq S_i \tag{6}$$

Equation 5 avoids time-wise overlap of any task pair on the same cluster. In the left-hand side of Equation 5, the $(1 - Y_{ij}) \times M$ term vanishes when $Y_{ij} = 1$. Otherwise, the constraint trivially satisfies due to the large constraint M. Similarly, if T_i starts after T_j, the term $(Y_{ij} \times M)$ in Equation 6 vanishes, and thus, the constraint gets satisfied. When T_j starts after T_i on the same cluster, then Y_{ij} becomes 1 and constraint Equation 5 enforces completion of T_i before T_j starts.

5) The total power consumption of the CMP must not exceed the power budget (Pow_BGT). Let Pow_{i,v,cl_c} represents the power consumption of the task T_i executing with version v at c^{th} cluster.

$$\sum_{i=1}^{n} \sum_{v=1}^{k_i} \sum_{cl_c \in \{C_L, C_H\}} Pow_{i,v,cl_c} \times Z_{i,v,cl_c} \leq Pow\_BGT \tag{7}$$

6) ***Objective:*** The objective of the formulation is to choose a feasible solution that maximises QoS:

$$Maximise \sum_{v=1}^{k_i} \sum_{i=1}^{n} \sum_{cl_c \in \{C_L, C_H\}} Z_{i,v,cl_c} \times O_i^v \tag{8}$$

3.1.1 *Offline Cache Management Considerations*

PRECIOUS-Offline primarily aims to generate efficient task schedules that maximise QoS while meeting timing and power constraints. Although offline cache management to optimise cache performance is a possible alternative, it faces fundamental limitations. The dynamic nature of runtime cache behaviour, influenced by data dependencies, inter-task interference, and unpredictable events, makes any static offline cache policy likely suboptimal. Hence, we propose a novel online technique, *PRECIOUS-Online*, to dynamically adapt to runtime cache behaviour.

3.2 *PRECIOUS-Offline* at work

Let us consider the real-time PTG according to Table 1 and Figure 2. We need to schedule this PTG on two clusters of processor cores (C_H and C_L), with a deadline $D_{PTG} =$ 142 time units, whereas frequencies of C_H and C_L have been assumed as 1.0 and 0.5, respectively. Our assumed power budget for both processors is set as $Pow\_BGT =$ 50. As per *PRECIOUS*'s constrained scheduling strategy, CPLEX [8], the ILP solver generates the scheduling output[1] shown in Figure 3. From Figure 3, it can be found that tasks T_1, T_2, T_3 and T_4 were executed with their highest versions, where T_1, T_2 and T_4 were scheduled at C_H and

1. Leveraging the typically small constant values for versions and cluster types, and the structured nature of PTGs, the *PRECIOUS-Offline* ILP formulation presents a computationally tractable approach for finding optimal schedules in an offline setting.

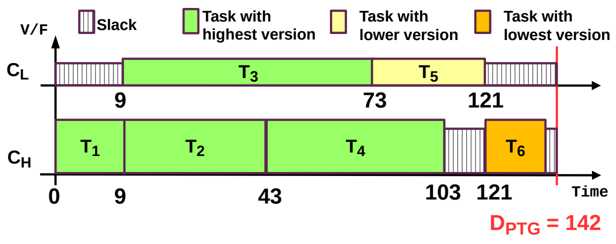

Fig. 3: Generated Task-Schedule (not to scale).

T_3 was scheduled at C_L. T_5 and T_6 execute at C_L and C_H, respectively, where T_6 has been scheduled with its lowest version (T_6^1) and T_5 has been scheduled with a lower version (T_5^2). Note that, our schedule satisfies the power constraint which is considered as $POW\_BGT = 50$. The entire PTG is able to finish by 140 time units and thus, $D_{PTG} = 142$ is met and total QoS value obtained is 45.

4 *PRECIOUS-Online*

In this section, we will illustrate the motivation behind devising *PRECIOUS-Online* mechanism for enhancing QoS along with the prime technique after discussing the basic concepts of MLC-MRAM and the cache organisation we will use in this work.

4.1 MLC-MRAM Cache: Background and Organisation

Magneto-resistive RAM (MRAM) employs magnetic tunneling junctions (MTJs) as storage elements, unlike SRAM caches that use electric charge. As illustrated in Figure 4, an MTJ consists of a free layer, a barrier (oxide), and a fixed reference layer. The free layer's magnetisation direction (MD) can be altered by a spin-polarised current [18]. Applying a current exceeding the critical switching current (I_C) from the free to the reference layer aligns their MDs (parallel state, low resistance, logic '0') or anti-aligns them (antiparallel state, high resistance, logic '1'). A cache bit storage cell in single-level cell (SLC) MRAM comprises an MTJ connected to an NMOS transistor (rightmost in Figure 4) [18], [22].

MLC-MRAM increases storage density by integrating multiple MTJs per cell. Parallel MLC-MRAM, which divides an MTJ's free layer into hard and soft domains for two bits [26], suffers from reliability issues due to process variations [46]. A more viable approach is series MLC-MRAM, stacking two MTJs [20]. As shown in the leftmost diagram of Figure 5, smaller and larger MTJs store soft and hard bits, respectively. Writing to series MLC-MRAM (middle of Figure 5) involves two steps: first, a large current ($I_{WH} > I_{C,Hard}$) switches both bits; second, a smaller current ($I_{C,Hard} > I_{WS} > I_{C,Soft}$) switches only the soft bit. Reading (shown in the right of Figure 5) also requires two sensing steps for soft and hard bits sequentially. Hard bits in MLC-MRAM exhibit higher write latency than soft bits due to complex programming and sensing [6], involving higher energy barriers and challenging distinction of magnetic states, thus balancing density with performance.

Two main design options exist for MLC-MRAM caches: Direct Mapping (DM) and Cell Split Mapping (CSM) [6]. DM straightforwardly divides each cache line into soft and hard bits, mapping N logic bits to $N/2$ MLC cells (e.g., a 512-bit line in 256 MLC cells, as in Figure 6), with half of each line's bits in soft bits and the other half in hard bits. Although simple and used in some prior designs [13], DM's two-step access incurs performance overhead by overlooking the faster access to soft bits. To address this, CSM, illustrated in Figure 6, evenly divides the cache way-wise into soft and hard bit regions, each holding an equal number of cache blocks. Access latency depends on a block's location in either region (soft or hard ways). Soft way hits are faster (one step), but hard way writes are more expensive due to the higher (write) latency of hard bits. Hence, an efficient data management mechanism is crucial to leverage MLC-MRAM's density while mitigating the higher write latency of hard bits. In *PRECIOUS*, we adopt CSM and will next discuss our intelligent data management technique for MLC-MRAM-based LLC, which enhances cache performance and, consequently, the overall system-level QoS for our approximate real-time tasks.

4.2 Preliminary analysis and Motivation

Based on our previous discussion and prior studies on MLC-MRAM it can be concluded that [6], [13], writing at the hard bits of MLC-MRAM is a costly operation in terms of energy as well as performance. In fact, energy usage while writing at the hard-bits will incur almost double energy than the soft bits, whereas write latency is also significantly higher in hard bits than soft ones. On the other hand, MLC-MRAM offers higher storage density over its SLC-MRAM counterpart, and our iso-area analysis through NVSim simulator [15] shows that the storage density almost doubles in MLC-MRAM over the SLC-MRAM. In *PRECIOUS*, by incorporating an iso-area MLC-MRAM with SLC-MRAM, we attempt to double the LLC capacity to improve the performance of the approximated real-time task-set so that overall QoS can be enhanced significantly. However, the advantage of larger LLC can only be exploited in MLC-MRAM if the costly write operations can be directed to the soft LLC ways.

Towards developing our novel write handling mechanism, we first analysed the read and write counts of our considered 8 and 5 benchmark applications from Ligra [37] and Cilk [38], respectively, where applications are based on hyper-graph and task-parallel based memory as well as compute intensive algorithms. The benchmarks are executed in an advanced version of gem5 [7], [42] by considering an HE CMP, where we have 60 In-Order (InO) energy efficient processor cores, which are accompanied by 4 high performance Out-of-Order (OoO) processor cores from RISC-V ISA. We concentrated on the LLC accesses for each application and plotted the percentages of read and write counts in Figure 7. This analysis shows the write percentage can be as high as 98%. At the same time, the average is around 51%, which indicates that if the majority of the write operations take place in the hard ways, higher access latency will be incurred, which might lead to severe performance aggravation. In fact, a higher write count on the hard ways will increase energy usage.

To study the behaviour of MLC-MRAM based LLC, we further analysed cache access pattern for an iso-area MLC-MRAM based LLC (8MiB, 16W) and compared it with an

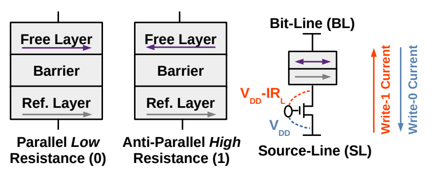

Fig. 4: Concept of MTJ and SLC-MRAM.

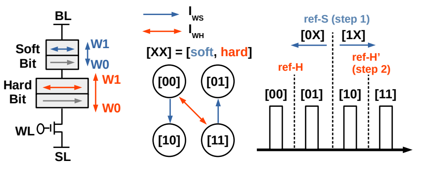

Fig. 5: Concept of MLC-MRAM.

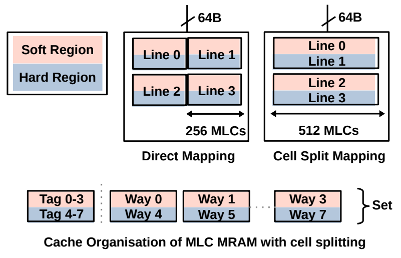

Fig. 6: MLC-MRAM based Cache.

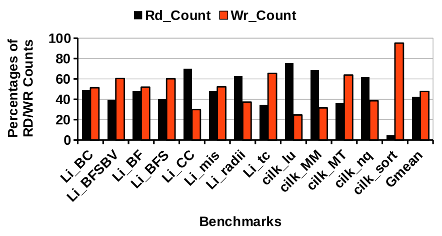

Fig. 7: Distribution of Read and Write operations.

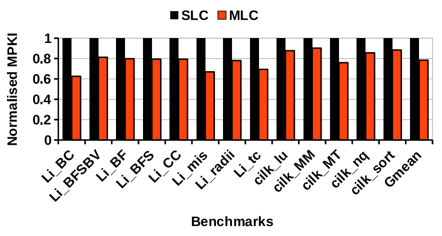

Fig. 8: Change in MPKI for iso-area SLC-MRAM and MLC-MRAM LLC.

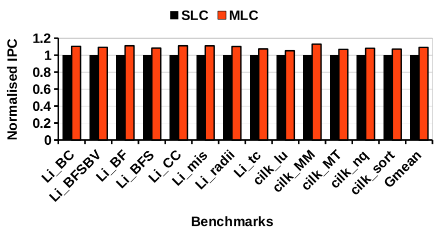

Fig. 9: Change in IPC for iso-area SLC-MRAM and MLC-MRAM LLC.

iso-area SLC-MRAM based LLC (4MiB, 16W), and plotted the changes in MPKI (misses per kilo instructions), IPC (instructions per cycle), and energy in Figure 8, 9 and 10, respectively. Doubling the LLC size by employing MLC-MRAM accumulates more live blocks on-chip that potentially lowers miss rates (MPKI) by 21% on average, which entails IPC improvement by 6 to 16% for all applications. This significant IPC improvement motivated us to employ iso-area MLC-MRAM for *PRECIOUS* to enhance performance; however, such performance improvement has been achieved at the cost of more than 50% increase in energy on average, caused by a substantial number of writes have been taken place at the hard ways. To ameliorate such significant energy usage, we propose a prudential LLC block management technique so that energy usage can be minimised without any noticeable performance loss.

4.3 Core Concept of *PRECIOUS-Online*

In this section, we will illustrate the core concept of *PRECIOUS-Online*, which reduces the write counts at the hard ways of the MLC-MRAM based LLC. As our LLC is attached to an HE system, we first statically redirect the requests to the soft and hard ways by restricting them as follows:

- $MLC\_In\_S\_Out\_H$ – requests generated from InO (OoO) cores will be handled at the soft (hard) ways;
- $MLC\_In\_H\_Out\_S$ – requests generated from InO (OoO) cores will be handled at the hard (soft) ways.

As write back operations are costlier at the hard bits, we first analysed the write counts at soft bit and hard bit regions. The total percentages of write back operations at $hard$ and $soft$ bit regions are plotted in Figure 11 for three different configurations: MLC (no restriction for cache access), $MLC\_In\_H\_Out\_S$ and $MLC\_In\_S\_Out\_H$, while considering 8 Ligra and 5 Clik applications. On average for MLC configuration, soft bit region serves 60% of the total writes, whereas, hard bit region serves the rest 40%. As write energy and latency are the prime bottlenecks for the MLC-MRAM based LLC and 40% of the entire write counts is substantially high, we further attempt to minimise the write counts at the hard bit regions. Hence, we next experimented with $MLC\_In\_S\_Out\_H$ and $MLC\_In\_H\_Out\_S$, and analysed the write back percentages for soft and hard bit regions. The analysis in Figure 11 depicts that, for our set of applications, the majority of the writes are requested by the InO cores, which are around 81% of the entire write counts. However, remaining 19% writes generated by the OoO cores are still considerable as it can potentially aggravate performance as well as energy efficiency at the hard bits. It is evident from our analysis that, a prudential write redirection needs to be implemented so that hard bit region will serve a minimal amount of write operations. In *PRECIOUS*, we next propose $MLC\_In\_S\_Out\_HS$, which will redirect all write operations of the InO cores to the soft bit regions, whereas the majority of write requests from the OoO cores will be redirected to the soft bit regions, however, a small portion of the write requests generated by OoO cores will still be served at the hard bit regions.

4.3.1 $MLC\_In\_S\_Out\_HS$: Managing LLC Blocks

The entire technique is illustrated in the flowchart shown in Figure 13. In this approach, we selectively redirect the allocation of OoO cores to the soft bit region using an additional bit named repl_bit. The repl_bit is added with every cache entry in the soft bit region as shown in Figure 12, which is used to preserve the residency of InO core blocks over the OOO core block by setting the repl_bit during the cache block allocation and cache block hit requested by the InO core. The resetting mechanism of repl_bit will be done on the LLC cache miss, which is illustrated in the flowchart given in Figure 13. Once an LLC miss is detected, the LLC controller will first inspect if the soft bit region has any invalid entry, as allocation in the soft bit region is always beneficial due to its low access latency and power consumption. However, once an invalid entry is detected at

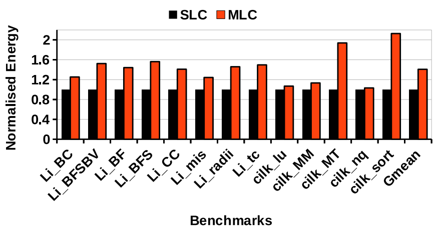

Fig. 10: Change in Energy for iso-area SLC-MRAM and MLC-MRAM LLC.

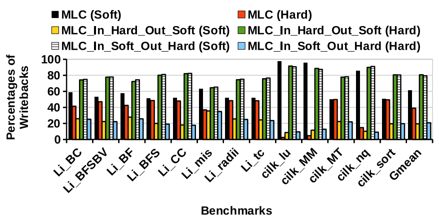

Fig. 11: Percentages of WBs for our primary LLC configurations.

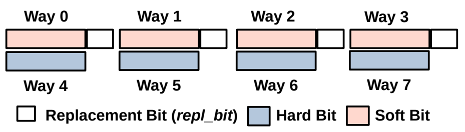

Fig. 12: Replacement Bits at the soft bit region.

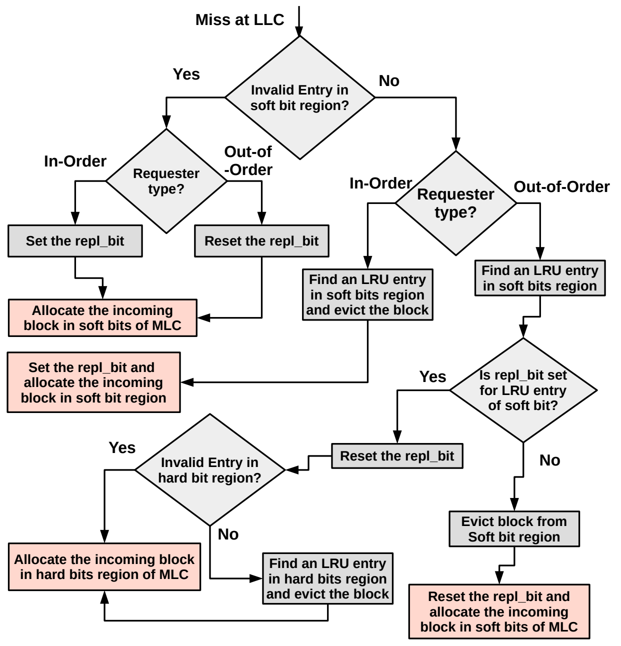
Miss at LLC
Invalid Entry in soft bit region?
Yes
No
Requester type?
In-Order
Out-of -Order
Set the repl_bit
Reset the repl_bit
Allocate the incoming block in soft bits of MLC
Set the repl_bit and allocate the incoming block in soft bit region
Requester type?
In-Order
Out-of-Order
Find an LRU entry in soft bits region and evict the block
Find an LRU entry in soft bits region
Is repl_bit set for LRU entry of soft bit?
Yes
No
Reset the repl_bit
Invalid Entry in hard bit region?
Yes
No
Allocate the incoming block in hard bits region of MLC
Find an LRU entry in hard bits region and evict the block
Evict block from Soft bit region
Reset the repl_bit and allocate the incoming block in soft bits of MLC

Fig. 13: Flow chart: $MLC\_In\_Soft\_Out\_HS$ policy for LLC Blocks.

the soft bit region, the requester type, i.e., an InO core or an OoO core, is next determined. If the requester is an InO core, the block will be allocated to the soft bit after setting the repl_bit. Otherwise, if the requester is an OoO core, the repl_bit will be reset, and the block will be allocated in the soft-bit region.

Once the soft bit region does not contain an invalid entry within an LLC set, the block requester (an InO or an OoO core) will be further determined. If the requester is an InO core, an LRU entry will be detected within the soft bit region, and a block will be allocated subsequently after setting the repl_bit. For a request from an OoO core, the LRU entry will be detected in the soft bit region at the beginning. Next, the respective repl_bit will be inspected, and if it is set, the controller will reset it and search for an invalid entry in the hard bit region. If the hard bit region has an invalid entry, the block will be allocated there, otherwise, the LRU entry from the hard bit region will be evicted, and the new block will be allocated in the hard bit region. However, an LRU entry will be evicted from the soft bit region in case the respective repl_bit is not set for this entry, and the block will be allocated subsequently after resetting the corresponding repl_bit. Note that the entire mechanism of *PRECIOUS-Online* will be implemented in the LLC controller of the CMP, and this will not incur any significant hardware overhead. Implementation of repl_bit at each entry of the soft bit region will incur less than 1% of area and cost overhead, but is effective enough to prioritise allocations and write requests from the InO over the OoO.

4.3.2 *Improving QoS & Energy Efficiency*

The entire mechanism of enhancing QoS and energy efficiency of *PRECIOUS-Online* is orchestrated by Algorithm 1. To balance per-task optimisation with the system level QoS enhancement goal, *PRECIOUS-Online* refines the task schedule from *PRECIOUS-Offline* (shown as the length of time-span between 0 to D_{PTG} in Figure 3 and referred to as $Frame$). This refinement involves making per-task decisions about optional part selection within the constraints of the offline schedule. At the beginning of each frame, our task-set and their individual starting time stamps and precedence are stored in the dispatch table. For individual clusters, the tasks are fetched in parallel from the dispatch table for execution as they have been scheduled by *PRECIOUS-Offline* (line 2 to 3). For a particular cluster, if the current task (T_i) is a source task, its execution will be started immediately (line 4 to 5). Note that, our underlying LLC hardware is implemented with MLC-MRAM technology, and by applying $MLC\_In\_S\_Out\_HS$, we have attempted to improve performance and reduce energy usage of the LLC. Hence, all of our tasks' execution length will be reduced, which can be exploited to improve overall system level QoS or energy efficiency.

The execution of non-source tasks can only be commenced if and only if all of its predecessors have been executed (line 7). During commencement of the execution, the M_i is fetched and executed at first, and subsequently scheduled version of the O_i is checked to determine whether the highest version is scheduled (line 9). If the highest version is not scheduled, our online mechanism will attempt to schedule a higher version. As incorporation of MLC-MRAM along with our novel block management policy reduces execution span of M_i for individual tasks, there might be room to dynamically schedule a higher version. Algorithm 1 will next evaluate the time left ($Max\_Time\_Left\_O_i$) to schedule a higher O_i. If the current task is a sink task, $Max\_Time\_Left\_O_i$ will have time till the deadline to schedule an apt O_i; otherwise, $Max\_Time\_Left\_O_i$ will be the time left till the earliest starting timestamp among all of its successors (line 9 to 14). Based upon the value assigned at $Max\_Time\_Left\_O_i$ and the versions of O_i, a suitable higher version will be executed next (line 15 to 18).

Algorithm 1: *PRECIOUS-Online*: Improving Energy Efficiency and QoS

```
Input: Dispatch Table, Set of $O_i$'s for all $T_i$'s, Deadline, $f_C$, $break\_even\_time$
1  for each Frame do
2      for each cluster in parallel do;
3      Get schedule details of each $T_i$ from the Dispatch Table and fetch it;
4      if $T_i$ is source task then
5          Execute $T_i$ at the predetermined cluster;
6      else
7          if All predecessors of $T_i$ has been executed then
8              Fetch $T_i$ and execute $M_i$;
9              if Highest $O_i$ is not scheduled then
10                 if $T_i$ is not the sink task then
11                     Get the earliest start time ($Early\_Start\_Succ\_T_i$) from $List\_Succ\_T_i$ (successor list of $T_i$, stored in dispatch table);
12                     $Max\_Time\_Left\_O_i = Early\_Start\_Succ\_T_i - Curr\_Time$;
13                 else
14                     $Max\_Time\_Left\_O_i = Deadline - Curr\_Time$;
15                 # Call the function that returns optional part with the highest possible accuracy which can run within $Max\_Time\_Left\_O_i$ ;
16                 $O_i$ = $get\_O_i(T_i, Max\_Time\_Left\_O_i)$ ;
17                 if $O_i$ is available then
18                     #Fetch the $O_i$ and start execution ;
19         if $T_i$ has been executed then
20             Get the start time of next task $T_j$ ($Start\_Time\_T_j$) scheduled on the same cluster;
21             $Est\_Slack\_Cycles = (Start\_Time\_T_j$ - $Curr\_Time)/f_C$;
22             if $Est\_Slack\_Cycles > break\_even\_time$ AND $\exists\ T_k \in Pred(T_j)$, $T_k$ is not yet executed then
23                 #Power gate the core ;
24                 while $Est\_Slack\_Cycles > 0$ do
25                     $Est\_Slack\_Cycles$--;
26                     if $\forall T_k \in Pred(T_j)$, $T_k$ has been executed OR $Est\_Slack\_Cycles == break\_even\_time$ then
27                         #Turn on the core;
28                         #break;
```

As our runtime policy ($MLC\_In\_S\_Out\_HS$) is also enabled during execution of O_i, there might be a chance that execution of O_i may finish earlier and a slack will be generated. Hence, once execution of an entire T_i is over, the starting time of the subsequent task (T_j) scheduled on the same processor is inspected, which is represented by $Start\_Time\_T_j$ (line 20). The cycles left (Est_Slack_Cycles) before $Start\_Time\_T_j$ are evaluated next by considering the current time-stamp ($Curr\_Time$) and the cluster frequency (f_C). A slack is only be declared at this point, if Est_Slack_Cycles is higher than the $break\_even\_time$ of the underlying core (provided by vendor), and the core will be power gated or turned off if any of the predecessors of the next task on the same cluster has not yet been executed. As $MLC\_In\_S\_Out\_HS$ dynamically reduces the execution length of individual tasks, the execution of a predecessor (T_k) of the next task (T_j) can be finished early, which can enable us to start execution of T_j before its actual schedule assigned offline. On the other hand, we also need to ensure that a cluster must have an amount of $break\_even\_time$ left before starting execution while deadline is guaranteed. Hence, turning on process for the cluster will be started if all predecessors (T_k) of the next task (T_j) on the same cluster have been executed, or it has $break\_even\_time$ left. The entire mechanism for cluster power saving is given in line 22 to 28. Finally, the improved QoS and reduced energy usage of the schedule can be determined from the executed O_i's and power gated time-spans, respectively.

5 Evaluation

The effectiveness of *PRECIOUS* relies on the synergy between *PRECIOUS-Offline* and *PRECIOUS-Online*: *PRECIOUS-Offline* generates the initial task schedule, while *PRECIOUS-Online* refines it to adapt to dynamic runtime conditions and optimise MLC-MRAM cache utilisation for improved QoS and energy efficiency. In this section, we evaluate *PRECIOUS-Offline* and *PRECIOUS-Online*, demonstrating how their combined operation enhances system-level QoS.

5.1 *PRECIOUS-Offline*: Evaluation Methodology

We define **N**ormalised **A**chieved **Q**oS (NAQ) as the ratio between the actually achieved QoS for the PTG, and the maximum achievable QoS by executing the highest versions of all tasks. NAQ can be formulated as: $NAQ = \frac{\sum_{i=1}^{n} QoS_i^j}{\sum_{i=1}^{n} QoS_i^{k_i}}$, where k_i represents the highest version of task T_i. Next, we model our multicore and the task-set:

- *Processor System:* An HE-multicore platform having 2 clusters (C_L (having 60 tiny InO cores) and C_H (having with 4 big OoO cores) of RISC-V cores has been considered. The power constraint of the individual cores is scaled and set as 10.5W based on the obtained runtime core power from McPAT [24].
- *Task-set:* Each PTG consists of a set of tasks (aka nodes) under precedence constraints and has a deadline D_{PTG}. Each task (T_i) is a multithreaded task (see Table 4), where all threads of a task are executed on the same cluster, characterised by execution times, ET_i. We assumed that a task can consume between 4×10^7 and 6×10^8 clock cycles [25], and they can have a maximum of 5 versions, i.e. $k = 5$. The assumptions regarding execution lengths also include memory cycles for our individual task consisting of Ligra, Cilk, and mix benchmark applications [37], [38] (Table 4). The heterogeneity factor γ_H and γ_L for individual tasks are chosen within the range of [0.4, 1] [25]. Having these parameters, execution times of task T_j on different clusters ($ET(j, cl)$) are determined, and we calculated the average execution requirement of a PTG (C_{PTG}^{avg}) of all tasks. The average utilization U_{PTG} of a PTG can be written as $\frac{C_{PTG}^{avg}}{D_{PTG}}$. In our experiments, C_{PTG}^{avg} is generated by keeping average utilisation of a PTG as 0.6. Given a set of values for task execution times, the total system workload (Sys_{WL}) can be obtained by:

$$Sys_{WL} = \frac{\sum_{j=1}^{n} \sum_{cl=1}^{|cl|} ET(j, cl)}{|n| \times |cl| \times C_{PTG}^{avg}} \quad (9)$$

- *Task Temporal Parameters:* For each T_i, based on which portion of the len_i is considered as mandatory part (M_i), the following cases are considered [16]: *(i)* $man\_low : M_i \sim U(0.2, 0.4) \times len_i$ (low portion of a task T_i's length (len_i) is for the mandatory part). *(ii)* $man\_med : M_i \sim U(0.4, 0.6) \times len_i$ (medium portion of a task T_i's length (len_i) is for the mandatory part). *(iii)* $man\_high : M_i \sim U(0.6, 0.8) \times len_i$ (high portion of a task T_i's length (len_i) is for the mandatory part).

5.2 *PRECIOUS-Offline*: Results and Analysis

Figure 14 depicts the NAQ achieved by *PRECIOUS-Offline* for different values of Sys_{WL} (system workloads). We observed that *PRECIOUS* is able to achieve upto 76% NAQ when Sys_{WL} is low. However, the QoS is reduced by 25% on average, when the workload is scaled up by 50%. Other insightful observations can also be derived from this figure. Firstly, as the system workload increases, the number of tasks in the PTG and the individual execution times of the task nodes increases (Equation 9) as the average utilisation of PTG is fixed, which eventually contributes to low NAQ values. This happens because higher the execution length of each task, lower the possibility of obtaining sufficient free slots in the scheduling period within the deadline, and thus, the probability of obtaining feasible schedules by selecting higher tasks' versions reduces.

We also compared our strategy with prior art, TD [25], HDA [31], and $DPMRS$ [35]. Towards a fair comparison with TD, we computed the overall energy budget based on the considered power constraint of the experimental framework of *PRECIOUS*. It can be observed that as execution demand for individual tasks goes up (due to an increase in Sys_{WL}), *PRECIOUS* maintains improved QoS by achieving a higher NAQ than TD. *PRECIOUS* is able to maintain 65% QoS at a 70% workload, for which TD achieves around 50% QoS. This is because the considered overall energy budget in TD would scale up with the higher Sys_{WL} as the number of tasks increases, and to obtain the feasible schedule, TD allows task migrations, which incur additional overhead and result in lower achieved QoS.

PRECIOUS-Offline achieves superior NAQ ($56 - 76\%$) over *HDA* ($38 - 51\%$) due to its globally optimal approach to task and version assignment for maximising optional computations. *HDA*, conversely, employs a heuristic for its initial task allocation (Master Problem) which may not be optimal, and critically, it terminates its iterative optimisation (between Master and Slave Problems) upon finding the first feasible solution. This premature termination restricts *HDA*'s exploration of the solution space for the most effective distribution of optional work. For *DPMRS* (achieving a NAQ of $20 - 42\%$), where *PRECIOUS-Offline* makes explicit choices about which version of a task to run and on which cluster type, considering the efficiency ($\gamma_{i,c}$) and power of that version on that cluster to maximise the overall NAQ. But, *DPMRS* selects processor frequencies (DVFS) and mappings to complete its workload with the least energy. Its DVFS is an energy-saving mechanism for a fixed workload, not a technique to create room for variable optional work.

Figure 15 illustrates the effect of processor heterogeneity on the system's QoS. We introduced the heterogeneity between the clusters by introducing the factor $\frac{\gamma_L}{\gamma_H}$. It can be observed that in the case of man_{high}, the increase in achieved NAQ is comparatively lower than that of man_{med} and man_{low}, while increasing the value of $\frac{\gamma_L}{\gamma_H}$. This can be attributed to the fact that *(i)* with $\frac{\gamma_L}{\gamma_H}$ increasing, the heterogeneity of the processors in clusters is reduced. As a result, the degree of skewness among the task execution times on two clusters decreases and the schedulability of the tasks increases, which contributes to higher QoS. *(ii)* Increase the mandatory portions of the individual tasks; lower the length of the optional portions. This results in the variance among the different versions of a task becoming smaller. Due to fewer variations among the optional parts of a task, there will be less impact on the achieved QoS. On the other hand, for man_{low}, we observe that the increase in NAQ is higher than the other two, and man_{med} offers a performance between man_{high} and man_{low}.

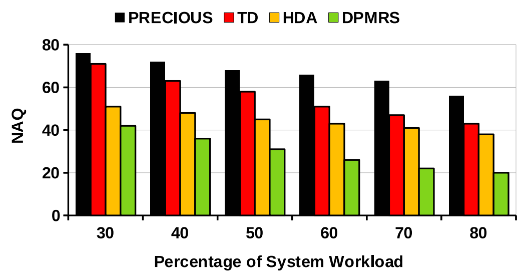

Fig. 14: Comparing NAQ of *PRECIOUS-Offline* vs. prior-art for different System-Workloads.

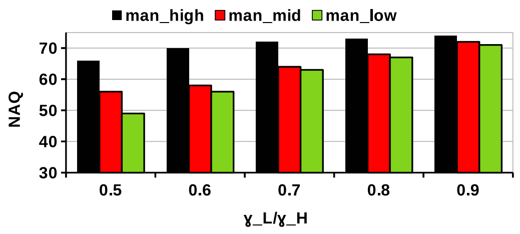

Fig. 15: Effect of processor heterogeneity on system's QoS.

5.3 *PRECIOUS-Online*: Simulation Infrastructure

PRECIOUS-Online has been evaluated by simulating the HE-multicore system in gem5 [7], with different core types, e.g., Big-core and Tiny-core, employing the setup proposed by Wang et al. [42]. Table 2 lists the system parameters used in our HE simulation. Note that, the set of four big-cores forms the C_H cluster, whereas the C_L cluster is the set of 60 tiny-cores. However, in our simulated system, each core has its own private L1 data and instruction caches, and all cores share the L2 cache, the LLC, which is considered a built-in MRAM. In our simulation, we configured our HE system with the two MRAM configurations: iso-area SLC-MRAM (*SLC*) of size 4 MiB and MLC-MRAM (*MLC*) of size 8 MiB. The baseline techniques, *MLC_In_S_Out_H* and *MLC_In_H_Out_S*, and the proposed technique *MLC_In_S_Out_HS* use the iso-capacity MLC-MRAM of 8 MiB. The timing and energy parameters of SLC and MLC-MRAM are given in Table 3, obtained from NVSim [15]. We used the Ligra and Cilk benchmark suite [37], [38] to construct task sets in approximate computing paradigm [41] for the performance evaluation. The applications chosen for evaluation are listed in Table 4.

TABLE 2: System parameters of *PRECIOUS-Online* [CC: clock cycle]

Parameters		Values
Tiny Core	ISA	RISC-V (RV64GC)
	Cores	60 cores (1.5GHz), single issue, in-order
	L1 Cache	4KiB L1I/D, 2-way, 1 CC
Big Core	ISA	RISC-V (RV64GC)
	Cores	4 cores (3GHz), 4-way out of order 16 entry LSQ, 128 entry ROB
	L1 Cache	64KiB L1I/D, 4-way, 1 CC
L2 Cache		Shared, 8-banks, 512KiB/1MiB SLC/MLC per bank, 8-way, 1 bank per mesh column
Network		8x8 mesh, XY routing, 16B per flit

TABLE 3: Timing and Energy Parameters of MRAMs.

Characteristics	SLC-STT	MLC-STT
Read Latency (in Cycles)	13	S:14 H:20
Write Latency (in Cycles)	49	S:50 H:95
Read Energy (in pJ)	415	S:427 H:579
Write Energy (in pJ)	876	S:1084 H:2653
Leakage Power	80.8 mW	

5.4 *PRECIOUS-Online* vs. Conventional MRAM-LLC

5.4.1 Performance Improvement and EDP Gains

We first evaluate the reduction in LLC misses (MPKI) and the IPC values for each benchmark application while using the following LLC configurations: *SLC*, *MLC*, *MLC_In_S_Out_H* and *MLC_In_H_Out_S*, and our proposed technique *MLC_In_S_Out_HS*. Using iso-area MLC with respect to SLC doubles the LLC size, which reduces the MPKI for all the benchmark applications, as a larger cache can accumulate more live blocks on-chip. Figure 16 depicts that, the MPKI for MLC has significantly been reduced up to 40%, which is around 19% on average, than SLC for all benchmarks.

In MLC, LLC blocks are allocated in a cache set without considering the underlying storage constructs, i.e., hard ways or soft ways. As writing or allocating a block in hard-way is costlier than the soft ways especially in terms of write energy and latency, the blocks need to be managed prudentially so that performance benefits by higher storage density of MLC can be exploited. Hence, we next considered *MLC_In_S_Out_H* and *MLC_In_H_Out_S* where all requests from a specific cluster will be handled either in soft region or in hard region. As block movement has been restricted for both of these configurations, individual clusters can only use half of the entire cache, which might lead to a higher miss rate within the partitioned LLC area, hence higher MPKI than MLC for all the benchmarks. However, in *MLC_In_H_Out_S*, a substantial amount of write accesses and allocations take place in the hard ways of the LLC, which incurs higher LLC access latency and energy, and that drops the IPC. On the other hand, *MLC_In_S_Out_H* experiences higher IPC, as many write accesses and allocations have been performed at the soft region. Basically, many InO cores in C_L generate a significant amount of LLC requests for almost all of our memory-intensive applications. Hence, handling such large write accesses and allocations at the hard region in $MLC\_In\_H\_Out\_S$ aggravates the overall energy efficiency and performance, which we further ameliorated in *MLC_In_S_Out_HS*.

In *MLC_In_S_Out_HS*, the higher write access and allocation counts generated by the C_L are handled at the soft region. In contrast, requests from OoO cores, i.e. C_H, will be handled in the soft region opportunistically, otherwise in the hard region. *MLC_In_S_Out_HS* although increases access pressure at the soft regions, which is, however, compensated by the energy savings at the hard region for all other three MLC configurations. We plotted the IPC changes for all of these configurations in Figure 17, where, for all the applications *MLC_In_S_Out_HS* shows 5.7% higher IPC, on average, than baseline SLC while IPCs of *MLC_In_S_Out_H* and *MLC_In_H_Out_S* drop by 1% and 9.6% than baseline SLC, respectively. To showcase the overall improvement in energy efficiency and performance, we next plot EDP in Figure 18. Due to higher write access and allocation counts, *MLC_In_H_Out_S* has higher energy usage, so the significantly higher EDP (94.2% higher, on average), whereas *MLC_In_S_Out_HS* shows almost the same EDP as the iso-area SLC. This implies that *MLC_In_S_Out_HS* has significantly lower energy usage while having higher performance. Note that, as our offline scheduling is based on the SLC configuration, employing MLC-MRAM LLC with *MLC_In_S_Out_HS* configuration entails performance improvements, that implicitly guarantees deadline with opportunity to enhance overall QoS.

Reliability Analysis: The potential impact of our cache management policy on the write distribution and the resulting reliability implications requires careful consideration. Figure 19 illustrates the percentages of writebacks and write energy consumption in the soft and hard bit regions. As shown, the *MLC_In_Soft_Out_HS* configuration directs a significantly larger portion of write requests to the soft-bit region (around 82% of writebacks), while these writes account for 65% of the total write-back energy. Although hard-bit writes are less frequent (18%), their higher energy consumption indicates that they still contribute a substantial 35% of the total write-back energy. This uneven distribution of both write counts and write energy coupled with thermal properties of underlying soft-bit and hard-bit circuitry could lead to differential heating and varying thermal profiles across the cache. Such localised thermal variations can negatively impact key reliability metrics like endurance and retention time, ultimately affecting the chip's lifetime. While a comprehensive reliability analysis, including detailed thermal modelling and simulation, is beyond the scope of this paper, we acknowledge the importance of this issue and identify it as a critical area for future research.

5.4.2 Impacts on Scheduling

The schedule shown in Figure 3 is next scaled by considering the tasks given in Table 4 along with runtime frequencies of the clusters and the respective magnitude of γ given in Table 1 and the online representation of Figure 3 is depicted in Figure 20**[A]**. By considering the performance

TABLE 4: Tasks formation with only Ligra [37], only Cilk [38] and mixing of both suites. For example, *BC (2)* implies two copies of *BC*, which is the same for others. Note that each copy runs with 64 threads. List of Benchmarks (names are prefixed with *Li_* for Ligra, and *cilk_* for Cilk applications): *Li_BC*, *Li_BFSBV*, *Li_BF*, *Li_BFS*, *Li_CC*, *Li_mis*, *Li_radii*, *Li_tc*, *cilk_lu*, *cilk_MM*, *cilk_MT*, *cilk_nq* and *cilk_sort*. The execution lengths (ELs) are in million cycles.

Tasks	Ligra Benchmarks (M_i, O_i)	Cilk Benchmarks (M_i, O_i)	Mixed Benchmarks (M_i, O_i)	EL ([M_i], [O_i])	Sel. Oi [EL]
T_1	*Li_BC (2), Li_BFSBV (2)*	*cilk_sort (2), cilk_nq (2)*	*Li_BC (2), cilk_lu (2)*	[80], [40]	#1 [40]
T_2	*Li_BF (2), Li_BFS (2)*	*cilk_MT (2), cilk_lu (2)*	*cilk_nq (2), Li_BFS (2)*	[200], [100, 160, 200]	#3 [200]
T_3	*Li_CC (2), Li_mis (2)*	*cilk_nq (2), cilk_MT (2)*	*Li_CC (2), cilk_MM (2)*	[200], [40, 80, 120]	#3 [120]
T_4	*Li_radii (2), Li_BFS (2)*	*cilk_lu (2), cilk_MM (2)*	*cilk_MT (2), Li_radii (2)*	[560], [320, 400]	#2 [400]
T_5	*Li_tc (2), Li_BFSBV (2)*	*cilk_nq (2), cilk_sort (2)*	*cilk_nq (2), cilk_sort (2)*	[160], [40, 80, 100]	#2 [80]
T_6	*Li_BFSBV (2), Li_radii (2)*	*cilk_MM (2), cilk_sort (2)*	*Li_BFSBV (2), cilk_MT (2)*	[200], [60, 120, 180]	#1 [60]

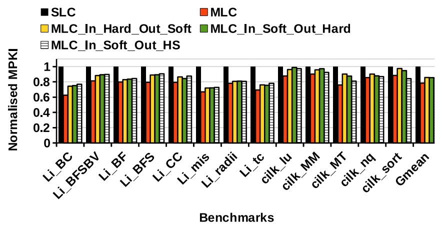

Fig. 16: Normalised MPKI.

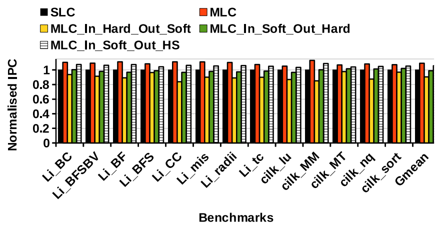

Fig. 17: Normalised IPC.

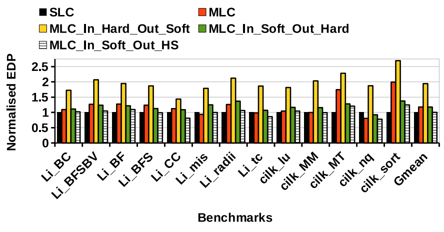

Fig. 18: Normalised EDP.

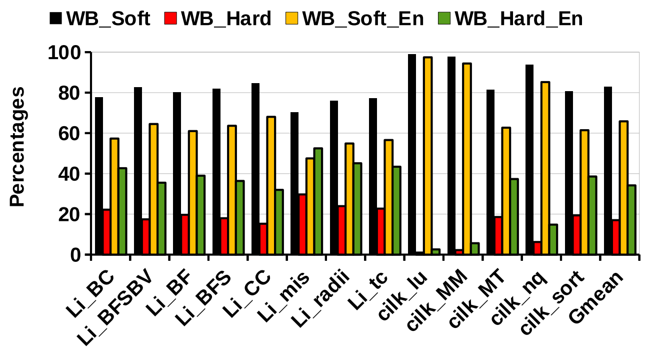

Fig. 19: Write Count (WB_Soft/Hard) and Write Energy (WB_Soft/Hard_En) distributions: soft vs. hard bits.

improvement of *MLC_In_S_Out_HS* for individual tasks (constructed with different combinations of benchmarks), reduced execution lengths and the dynamic slacks are derived, and the respective schedule is shown in Figure 20**[B]**. However, these dynamic slacks are next exploited opportunistically by Algorithm 1 to improve system-wide QoS by executing higher versions of T_5 and T_6, where the rest of the tasks were scheduled with their highest versions.

By employing Algorithm 1, the tasks have been given the privilege of early start and completion of their execution without violating the precedence and timing constraints and offer ample room to improve power and QoS. Finally, Algorithm 1 can execute T_5 with its highest version, while T_6 has been executed with a higher version, but not the highest, while initially scheduled with its lowest version by *PRECIOUS-Offline*. Additionally, Algorithm 1 further exploits the slacks at both the clusters to save energy through power gating the clusters. The final schedule after employing Algorithm 1 is shown in Figure 20**[C]**, and the respective energy savings for three schedules (**[A]**, **[B]** and

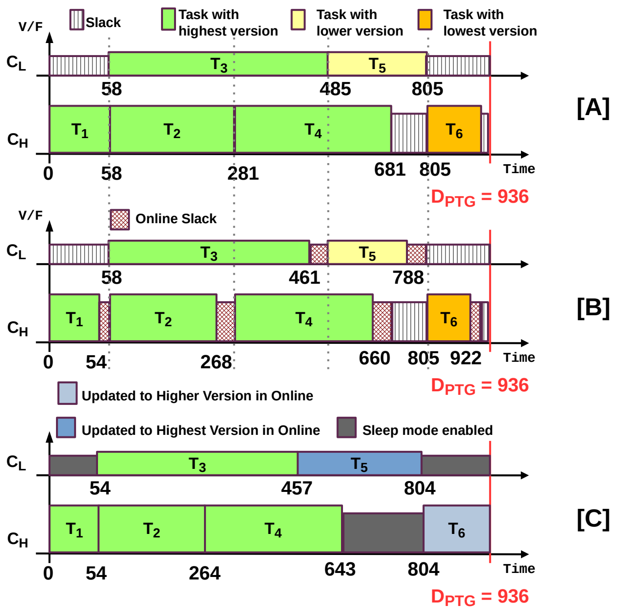

Fig. 20: QoS Update and Energy Savings by *PRECIOUS-Online*.

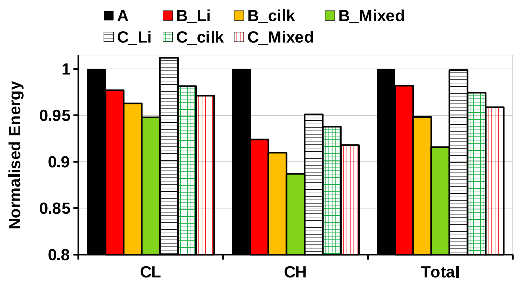

Fig. 21: Energy Usage by *PRECIOUS-Online*.

[C]) are shown in Figure 21. We have shown the change in energy for both clusters while executing with **[B]** and **[C]**. For **[B]**, energy saving is more than 5% for both clusters, which is partially traded off in **[C]** to improve QoS; however, **[C]** still has the same energy usage as **[A]**. Note that, due to executing the highest version of T_5 at C_L, the energy usage in **[C]** is almost the same as the **[A]**, however turning off C_H during the online slack between T_4 and T_6 saves energy around 4% at C_H. As the number of cores in C_L is much higher than the core counts in C_H, the overall energy saving is following the trend shown for C_L. However, reducing power density at high performance cores in C_H will lead to improved thermal safety. The overall QoS improvement is 9.0%, shown in Table 5, with the updated task-versions.

TABLE 5: Outputs of *PRECIOUS-Offline* and *Online* for the task-set constructed with Ligra (Table 4)

Tasks	Mapped Cluster	Scheduled Version (Offline)	Updated Version (Online)
T_1	C_H	1	1
T_2	C_H	3	3
T_3	C_L	3	3
T_4	C_H	2	2
T_5	C_L	2	3
T_6	C_H	1	2
Improvement in Achieved QoS			9.0%

Core Utilisation Profiles: To provide a comprehensive understanding of the system behaviour, we analysed the utilisation profiles of the InO and OoO cores during our experiments. OoO cores (in C_H) are primarily responsible for executing tasks T_1, T_2, T_4, and T_6. This leads to periods of high activity, especially during the execution of time-critical task components. InO cores (in C_L), while handling tasks T_3 and T_5, contribute the majority of the write requests (76%). *PRECIOUS-Online* leverages the dynamic slack generated by improved task execution to exploit the higher versions of tasks. The cache management strategy is designed to accommodate these distinct profiles: prioritising soft region access for InO cores to minimise write overhead and providing opportunistic soft region access for OoO cores to support performance.

6 Relevant Literature

Based on problem objectives, the scheduling can be broadly categorised as energy-aware and QoS-aware. In case of energy-aware task deployment, energy usage is minimised while considering system-wide constraints [5], [36]. For homogeneous systems, DVFS and DPM are primarily considered enhancing energy efficiency [17]. The recent shift from homogeneous to HE platform introduces additional research challenges, such as extra variables regarding the core selection and voltage/frequency (V/F). Even scheduling becomes more challenging in such HE paradigms [25] that integrate a set of high-performance, larger processor cores with a number of power-efficient smaller cores [3]. Researchers attempted to develop HE energy-aware schedulers for real-time tasks considering system's constraints [35]. Another recent study [32] enables virtualisation on heterogeneous MPSoCs by generalising real-time static partitioning for virtual machines across different ISAs, ensuring spatio-temporal isolation. *SCENIC* [12] presents an end-to-end co-design approach using a control capability function to efficiently design and schedule intelligent control tasks on heterogeneous computing architectures, optimising for latency, performance, and reliability.

In [9], the concept of the imprecise computing (IC) or approximate computing task of a real-time system towards maintaining an energy budget was introduced. Other earlier efforts [10], [30] that explored approximate task scheduling for time-critical systems for maximising QoS while trimming energy usage, considered homogeneous computing paradigms. On the other hand, a few works also focused on HE platforms [25], [43]. To enhance the QoS with energy and time constraints, a DVFS-based approach was devised [30]. However, the considered tasks were independent. In another work [39], the authors considered dependent tasks, and optimised frequency and task assignment to maximise the system QoS under real-time energy budget.

To reduce leakage power and area occupancy of the conventional SRAM-based LLC, researchers have been exploring alternative emerging memory technologies over the decades [40]. Prior research on MRAM primarily focused on the higher write latency and energy. For example, to alleviate the adverse impact of costly write operations through read-preemptive write buffer, a novel technique was proposed [45]. Relocating and managing the frequently used LLC blocks over multiple retention zones is a way in practice to boost system performance and has been utilised in MRAM cache design [1]. However, since its presentation [26], MLC MRAM has also gained attention for its density benefit [13]. But, MLC MRAMs inevitably suffer from the long read/write latency and energy, especially on accessing hard bits [20]. Jiang et al. presented line-paring mechanism that partitions the parallel MLC into read-fast-write-slow and write-fast-read-slow regions [21] to improve energy of MLC-MRAM caches.

Sampaio et al. proposed an approximation-aware MLC STT-RAM cache architecture, that partially protects cache from reliability issues and, in turn, leverages variable resilience of applications to adaptively trim protection overhead within a specified error threshold [34]. The reliability of data storage in the hard region has further been improved by implementing a "two-step elimination" protocol at the MLC-MRAM based LLC [19]. In contrast, a read error resilient MLC-MRAM has been proposed to improve reliability by mitigating both sensing and disturbance errors at the soft regions [44]. However, all of these prior techniques were explicitly proposed for the shared LLC in the homogeneous CMPs, whereas incorporating MLC-MRAM based shared LLC in HE environment needs novel solution to manage the cache accesses so that performance per watt of the LLC can be improved while preserving the performance benefits offered by the HE cores.

***PRECIOUS* over Prior Art:** In *PRECIOUS*, we investigated the potential of MLC-MRAM based LLC in improving performance energy efficiency of an approximated HE real-time computing paradigm. Basically, *PRECIOUS* first introduces a novel HE-offline scheduling algorithm for a dependent approximate real-time task-set, intending to improve the QoS (Sec. 3). The QoS is further improved by exploiting

TABLE 6: Summary: *PRECIOUS* vs. Prior Art

Prior Art	Heterogeneous	Dependent Task	Approx. Comp.	Online Adapt.	Cache Aware	QoS Aware	Energy Aware
TD [25]	✓	✓	✓	×	×	✓	×
Mo et al. [31]	✓	✓	✓	×	×	✓	×
DPMRS [35]	✓	✓	×	×	×	×	✓
Wei et al. [43]	✓	✓	✓	✓	×	✓	✓
Cao et al. [9]	✓	×	✓	✓	×	✓	✓
Aydin et al. [4]	×	✓	✓	×	×	✓	×
Prepare [10]	×	✓	✓	✓	×	✓	✓
Mo et al. [30]	✓	✓	✓	×	×	✓	✓
ACCURATE [33]	×	✓	✓	✓	✓	✓	✓
PRECIOUS	✓	✓	✓	✓	✓	✓	✓

storage benefits of MLC-MRAM based LLC whereas higher write energy and latency of the MLC-MRAM based LLC is further mitigated by our novel online technique (Sec. 4) that opportunistically manages the frequently used write intensive blocks at the soft region. The task-level performance gain makes room for improving QoS by running higher task-version or energy efficiency by turning off the cores during slacks. Our results (in Sec. 5, as evident from Table 5) exhibit the efficacy of the offline and online techniques of *PRECIOUS*, and specifically our offline scheduler outperforms prior art [31].

To the best of our knowledge, *PRECIOUS is the first technique that prudentially exploits the storage advantage of MLC-MRAM based LLC over conventional SLC-MRAM based LLC while ameliorating its write related issues to enhance QoS gained by its offline HE scheduler targeting an approximate dependent time-critical task-set without violating system-wide constraints.* We further summarised the differences between the prior approaches and *PRECIOUS* in Table 6.

7 Conclusions

In *PRECIOUS*, we introduce a novel hybrid offline-online *scheduling strategy* for approximate real-time tasks in an HE multiprocessor platform equipped with MLC-MRAM based LLC. *PRECIOUS-Offline* generates a schedule for a dependent task-set with an objective to maximise the QoS while considering other system-wide constraints. *PRECIOUS-Online* further improves the tasks' performance by prudentially leveraging MLC-MRAM storage density and enhancing system level QoS by considering the tasks scheduled with lower accuracy and opportunistically scheduling the higher task versions without incurring any energy overhead. In addition, *PRECIOUS-Online* saves energy by turning off the processor cores during the slacks generated at runtime. *PRECIOUS* is the first exploration that studied scheduling for a dependent approximate time-critical task-set on an HE-computing platform that exploits higher storage density of MLC-MRAM. Overall, *PRECIOUS*'s effectiveness stems from its hybrid offline-online approach, where *PRECIOUS-Offline* generates a static schedule, and *PRECIOUS-Online* dynamically adapts and optimises it for MLC-MRAM characteristics, demonstrating the necessity of both phases for achieving high QoS and energy efficiency. *PRECIOUS-Offline* achieves up to 76% QoS for a specific task-set, surpassing prior art, whereas *PRECIOUS-Online* enhances QoS by 9.0% by reducing cache miss-rate by 19% on a 64-core HE system without incurring any energy overhead over conventional MRAM cache. In future, we plan to extend *PRECIOUS* to support concurrent task execution within clusters by transitioning to a task-to-core mapping. Furthermore, incorporating DVFS and task migration strategies could further enhance the framework's performance and energy efficiency.

Acknowledgments

This work is supported by the UK Engineering and Physical Sciences Research Council (EPSRC) through grants-$EP/Z533749/1$ and $c-14456642$ (APRIL Hub Seed Fund). For the purpose of open access, the author has applied a Creative Commons Attribution (CC BY) licence to any Author Accepted Manuscript version arising.

References

[1] S. Agarwal *et al.*, "Architecting Selective Refresh based Multi-Retention Cache for Heterogeneous System (ARMOUR)," in *DAC*, 2023.

[2] ——, "TEEMO: Temperature aware energy efficient multi-retention stt-ram cache architecture," in *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2024, pp. 852–864.

[3] M. Ansari *et al.*, "Thermal-aware standby-sparing technique on heterogeneous real-time embedded systems," *IEEE TETC*, 2021.

[4] H. Aydin *et al.*, "Optimal reward-based scheduling for periodic real-time tasks," *IEEE TC*, 2001.

[5] A. Bhuiyan *et al.*, "Energy-efficient parallel real-time scheduling on clustered multi-core," *IEEE TPDS*, 2020.

[6] X. Bi *et al.*, "Unleashing the potential of MLC STT-RAM caches," in *ICCAD*, 2013.

[7] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH CAN*, 2011.

[8] C. Bliek *et al.*, "Solving mixed-integer quadratic programming problems with IBM-CPLEX: a progress report," in *RAMP*, 2014.

[9] K. Cao *et al.*, "QoS-adaptive approximate real-time computation for mobility-aware IoT lifetime optimization," *IEEE TCAD*, 2019.

[10] S. Chakraborty *et al.*, "Prepare: Power-Aware Approximate Real-Time Task Scheduling for Energy-Adaptive QoS Maximization," *ACM TECS*, 2021.

[11] S. Chakraborty and M. Själander, "WaFFLe: Gated cache-ways with per-core fine-grained DVFS for reduced on-chip temperature and leakage consumption," *ACM TACO*, 2021.

[12] J. Chen *et al.*, "SCENIC: Capability and Scheduling Co-Design for Intelligent Controller on Heterogeneous Platforms," in *RTSS*, 2024.

[13] Y. Chen *et al.*, "Access scheme of multi-level cell spin-transfer torque random access memory and its optimization," in *MWSCAS*, 2010.

[14] H. Djigal *et al.*, "Task scheduling for heterogeneous computing using a predict cost matrix," in *Workshop Proceedings of ICPP*, 2019.

[15] X. Dong *et al.*, "NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory," *IEEE TCAD*, 2012.

[16] A. Esmaili *et al.*, "Energy-aware scheduling of task graphs with imprecise computations and end-to-end deadlines," *ACM TODAES*, 2019.

[17] ——, "Modeling processor idle times in MPSoC platforms to enable integrated DPM, DVFS, and task scheduling subject to a hard deadline," in *ASP-DAC*, 2019.

[18] M. Hosomi *et al.*, "A novel nonvolatile memory with spin torque transfer magnetization switching: Spin-RAM," in *IEEE IEDM*, 2005.

[19] J.-W. Hsieh *et al.*, "TSE: Two-Step Elimination for MLC STT-RAM Last-Level Cache," *IEEE TC*, 2021.

[20] T. Ishigaki *et al.*, "A multi-level-cell spin-transfer torque memory with series-stacked magnetotunnel junctions," in *Symposium on VLSI Technology*, 2010.

[21] L. Jiang *et al.*, "Constructing large and fast multi-level cell STT-MRAM based cache for embedded processors," in *DAC*, 2012.

[22] T. Kawahara *et al.*, "2 Mb SPRAM (SPin-Transfer Torque RAM) With Bit-by-Bit Bi-Directional Current Write and Parallelizing-Direction Current Read," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 1, 2008.

[23] D. Li and J. Wu, "Minimizing energy consumption for frame-based tasks on heterogeneous multiprocessor platforms," *IEEE TPDS*, 2014.

[24] S. Li *et al.*, "McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *MICRO*, 2009.

[25] X. Li *et al.*, "Approximation-aware task deployment on heterogeneous multi-core platforms with DVFS," *IEEE TCAD*, 2022.

[26] X. Lou *et al.*, "Demonstration of multilevel cell spin transfer switching in MgO magnetic tunnel junctions," *Applied Physics Letters*, 2008.

[27] Y. meng Chen *et al.*, "A scheduling algorithm for heterogeneous computing systems by edge cover queue," *Knowledge-Based Systems, Elsevier*, 2023.

[28] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, 2016.

[29] ——, "A survey of techniques for cache locking," *ACM TODAES*, 2016.

[30] L. Mo *et al.*, "Energy-quality-time optimized task mapping on DVFS-enabled multicores," *IEEE TCAD*, 2018.

[31] ——, "Approximation-aware task deployment on asymmetric multicore processors," in *DATE*, 2019.

[32] D. Ottaviano *et al.*, "The Omnivisor: A Real-Time Static Partitioning Hypervisor Extension for Heterogeneous Core Virtualization over MPSoCs," in *ECRTS*, 2024.

[33] S. Saha *et al.*, "ACCURATE: Accuracy maximization for real-time multi-core systems with energy efficient way-sharing caches," *IEEE TCAD*, 2022.

[34] F. Sampaio *et al.*, "Approximation-aware Multi-Level Cells STT-RAM cache architecture," in *CASES*, 2015.

[35] D. Senapati *et al.*, "Energy-aware real-time scheduling of multiple periodic dags on heterogeneous systems," *IEEE TCAD*, 2023.

[36] S. Z. Sheikh and M. A. Pasha, "Energy-efficient real-time scheduling on multicores: A novel approach to model cache contention," *ACM TECS*, 2020.

[37] J. Shun and G. E. Blelloch, "Ligra: A lightweight graph processing framework for shared memory," in *PPPoPP*, 2013.

[38] J. Shun *et al.*, "Brief announcement: The problem based benchmark suite," in *ACM SPAA*, 2012.

[39] G. L. Stavrinides and H. D. Karatza, "An energy-efficient, QoS-aware and cost-effective scheduling approach for real-time workflow applications in cloud computing systems utilizing DVFS and approximate computations," *Future Generation Computer Systems*, vol. 96, pp. 216–226, 2019.

[40] G. Sun *et al.*, "A novel architecture of the 3D stacked MRAM L2 cache for CMPs," in *HPCA*, 2009.

[41] K. Tovletoglou *et al.*, "HaRMony: Heterogeneous-Reliability Memory and QoS-Aware Energy Management on Virtualized Servers," in *ASPLOS*, 2020.

[42] M. Wang *et al.*, "Efficiently supporting dynamic task parallelism on heterogeneous cache-coherent systems," in *ISCA*, 2020.

[43] T. Wei *et al.*, "Cost-constrained QoS optimization for approximate computation real-time tasks in heterogeneous MPSoCs," *IEEE TCAD*, 2018.

[44] W. Wen *et al.*, "Read Error Resilient MLC STT-MRAM Based Last Level Cache," in *ICCD*, 2017.

[45] X. Wu *et al.*, "Hybrid cache architecture with disparate memory technologies," in *ISCA*, 2009.

[46] Y. Zhang *et al.*, "Multi-level cell STT-RAM: Is it realistic or just a dream?" in *ICCAD*, 2012.

Sangeet Saha is a Lecturer in Computer Science at the University of Essex, UK. From 2018 to 2021, he was a Senior Research Officer at the EPSRC National Centre for Nuclear Robotics, based at the University of Essex. He received the YERUN Research Mobility Award in 2021. His research focuses on real-time scheduling, reconfigurable computing, embedded systems, and cloud computing. He has published extensively in conferences like CODES+ISSS, ISCAS, and journals like ACM TECS, IEEE TCAD, etc.

Shounak Chakraborty (Senior member, IEEE) is currently working as an Assistant Professor in the Department of Computer Science, Durham University, UK. His research interests include high performance computer architectures, non-volatile memory technologies, on-chip thermal management, and compilers. Prior to joining NTNU, Shounak obtained his PhD in Computer Science and Engineering from IIT Guwahati, India in February 2018, and also worked as assistant professor at IIIT Guwahati, India.

Sukarn Agarwal is currenlty working as an Assistant Professor in Electrical Engineering and Computer Science Department at the Indian Institute of Science Education and Research, Bhopal India. He earned his PhD degree in Computer Science and Engineering from IIT Guwahati, India, in March 2020. His research interests include Emerging Memory Technologies, Memory System Design, Network-on-Chip design, and Thermal Aware Chip Management. He published many of his research contributions in conferences like DAC, PLDI, ASAP, etc. and also published several of his research outcomes in journals like IEEE TVLSI, ACM TECS, IEEE TC, etc.

Magnus Själander is working as a Professor at the Norwegian University of Science and Technology (NTNU). He obtained his Ph.D. from Chalmers University of Technology in 2008. Before joining NTNU in 2016 he has been a researcher at Chalmers University of Technology, Florida State University, and Uppsala University. Själander's research interests include hardware/software co-design (compiler, architecture, and hardware implementation) for high-efficiency computing.

Klaus McDonald-Maier is currently the Head of the Embedded and Intelligent Systems Laboratory and Director Research, University of Essex, Colchester, U.K. He is also the founder of UltraSoC Technologies Ltd., the CEO of Metrarc Ltd., and a Visiting Professor with the University of Kent. His current research interests include embedded systems and system-on-chip design, security, development support and technology, parallel and energy-efficient architectures, computer vision, data analytics, and the application of soft computing and image processing techniques for real-world problems. He is a member of VDE and a Fellow of the BCS and IET.